Types of local consistency

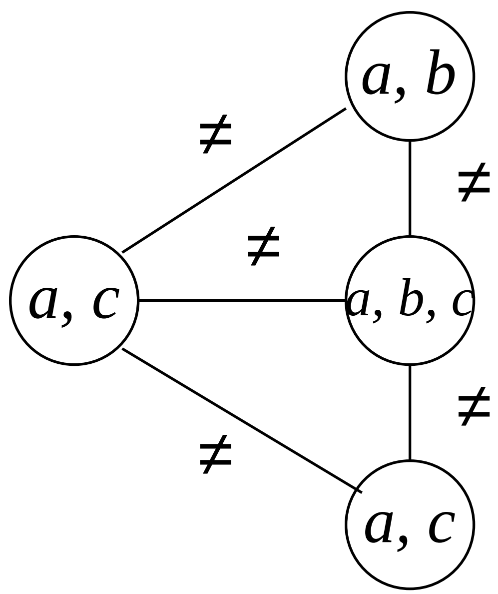
- (i, j)-consistency
  - any solution to a subproblem of i variables can be extended to a solution including any j additional variables
- k-consistency
  - any solution to a subproblem of $k-1$ variables can be extended to a solution including an additional variable
  - equivalent to (k, 1)-consistency
  - arc consistency is 2-consistency
  - path consistency is 3-consistency
- (1, k)-consistency is k inverse consistency

Time and space complexity

- Both k–consistency and k inverse consistency take time exponential in k
- In general, k-consistency requires creating and storing constraints involving $k-1$ variables
  – can require $O(d^{k-1})$ space
- However, k inverse consistency only filters out values for variables
  – worst case space requirement is linear
  – …and can even decrease the space requirement

Neighborhood inverse consistency

- Let the *neighborhood* of a variable *v* consist of the variables with which *v* shares a constraint
- Neighborhood inverse consistency enforces for each variable *v* *k* inverse consistency for *v* and its *k–1* neighbors



Neighborhood inverse consistency

function *NIC*

Insert each variable v onto agenda A

while $A \neq \{\}$ **do**

$v = pop(A)$ and $deleted = false$

for each a in $dom(v)$ **do**

if there is no solution to $Nbd(v)$ with v assigned a **then**

$dom(v) = dom(v) \setminus \{a\}$ and $deleted = true$

if $dom(v) = \{\}$ **then** **return** "domain wipeout"

endif

if *deleted* **then**

for each u in $Nbd(v)$ **do** $A = A \quad \{u\}$

endwhile

return "consistent"

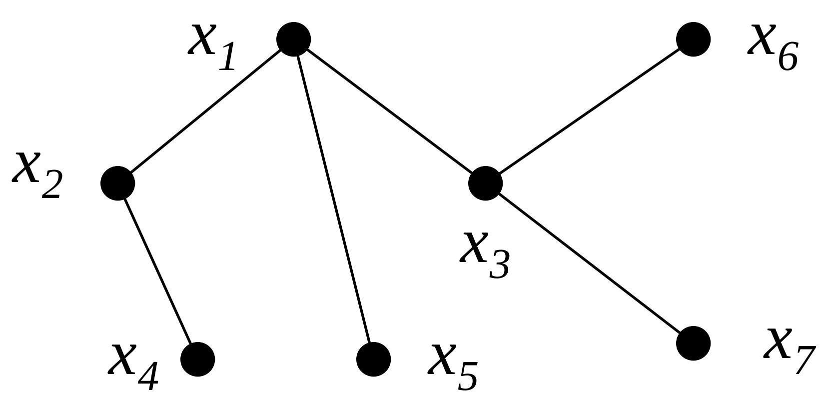
end *NIC*

Constraint graphs

- The *primal-constraint graph* of a constraint network has
  - a node for each variable
  - an undirected edge between two nodes if the corresponding variables occur in the same constraint
- The *dual-constraint graph* of a constraint network has
  - a node for ecah constraint
  - an undirected labeled edge between two nodes that share a common variable
  - edges are labeled by the shared variables

Tree networks

- Constraint networks whose primal graph is a tree
- Can be solved in time linear in the number of variables



Solving a tree network

procedure *tree-algorithm*

Generate a *rooted tree* ordering, $d = x_1, x_2, \ldots, x_n$

for $i = n$ **downto** 1 **do**

$revise(x_{p(i)}, x_i)$

if $dom(x_{p(i)}) = \{\}$ **then** "no solution exists"

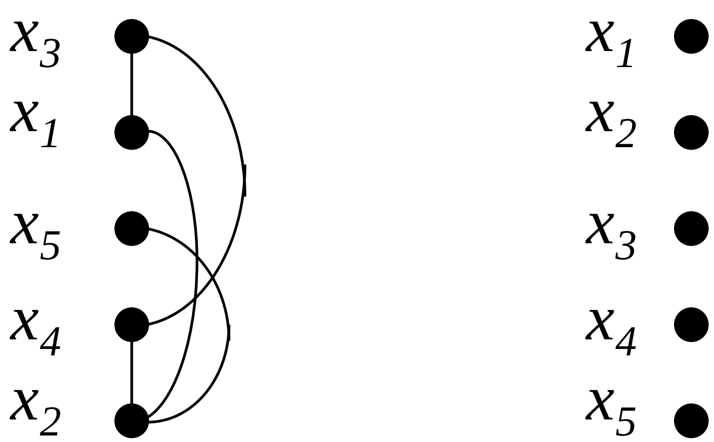
endfor

Use backtracking to instantiate variables along d

end *tree-algorithm*

Width

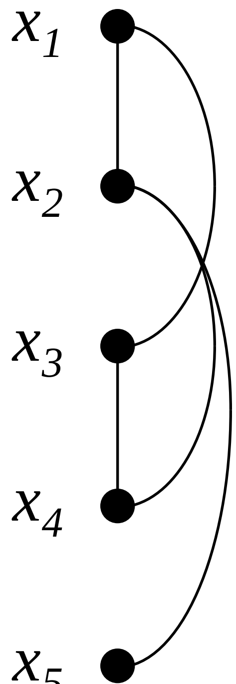
- Given a variable ordering $d = x_1, x_2, \ldots, x_n$
  - the *width of a node* x_i is the number of edges that connect x_i to nodes *earlier* in the ordering
- The *width of an ordering* is the maximum width of all nodes
- The *width of a graph* is the minimum width of all orderings



Directional consistency

- Given an ordering *d*, *directional i-consistency* along *d* requires that any consistent instantiation of *i–1* variables can be consistently extended by any variable that *succeeds all of them* in the ordering *d*
  - *strong* directional *i* consistency also requires directional *j* consistency for all $j < i$
- **Theorem:** An ordered constraint graph is *backtrack free* if the level of directional strong consistency along the order is greater than the width of the ordering

Enforcing directional consistency



Adaptive consistency

procedure *Adaptive-consistency*

for $i = n$ **downto** *1* **do**

Connect all elements in *parents*(x_i)

Perform *consistency*(x_i, *parents*(x_i))

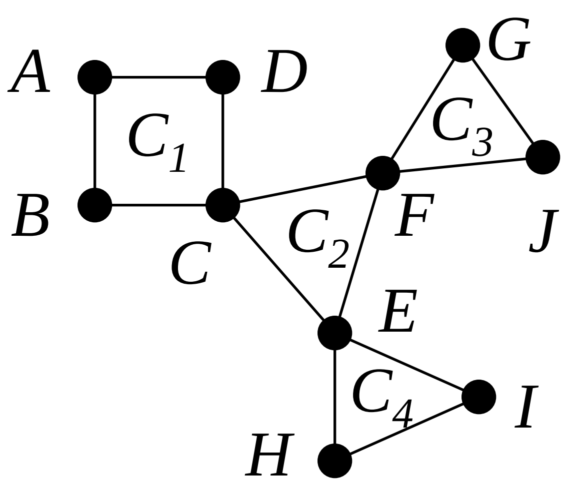
endfor

end *Adaptive-consistency*

- Topology of *induced* graph can be found *a priori*
- Let $w^*(d)$ be the width of induced graph

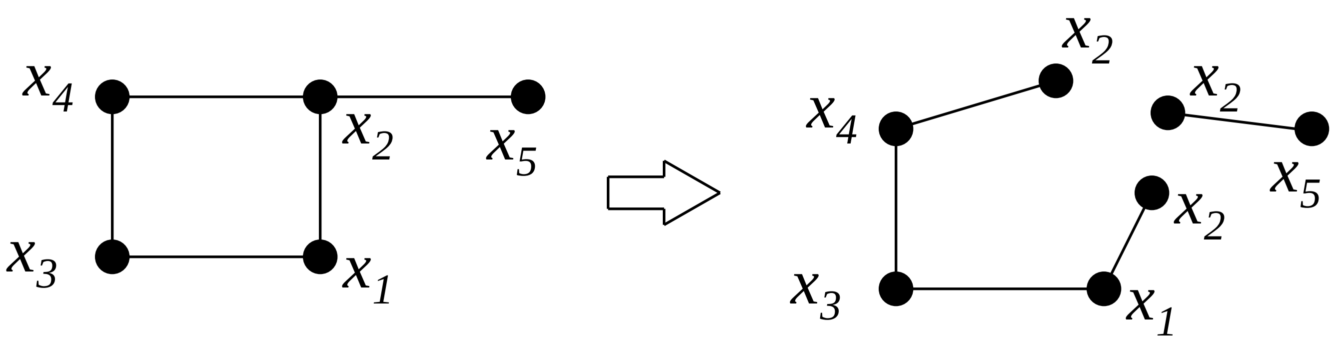
Nonseparable components

- *Separation* nodes (or *articulation* nodes) separate the graph into *nonseparable* components (or *biconnected* components)



Cycle cutset scheme

- Instantiating a variable cuts its own cycles



- When a group of instantiated variables constitutes a cycle cutset, the remaining network can be solved using the *tree algorithm*